"A good decision is based on knowledge and not on numbers." – Plato

"Once you make a decision, the universe conspires to make it happen." – Ralph Waldo Emerson

"The quality of decision is like the well-timed swoop of a falcon which enables it to strike and destroy its victim." – Sun Tzu

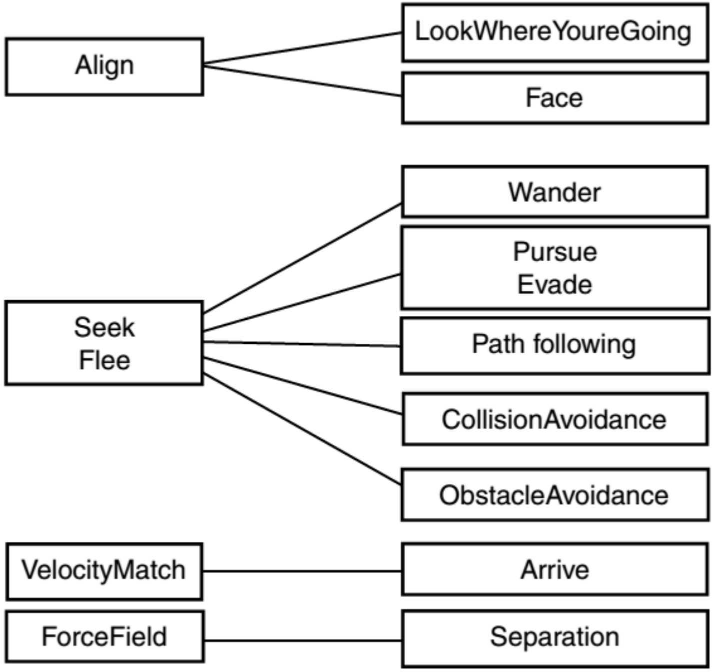
- Wikipedia: "[Perfect is the enemy of good]"
- "Everything should be as simple as possible, but not simpler." – Einstein
- Occam (of Razor fame – parsimony, economy, succinctness in logic/problem-solving)
  - "Entities should not be multiplied more than necessary"
  - "Of two competing theories or explanations, all other things being equal, the simpler one is to be preferred."
- "All that is complex is not useful. All that is useful is simple." – Mikhail Kalashnikov (of AK-47 fame)

Class N-4

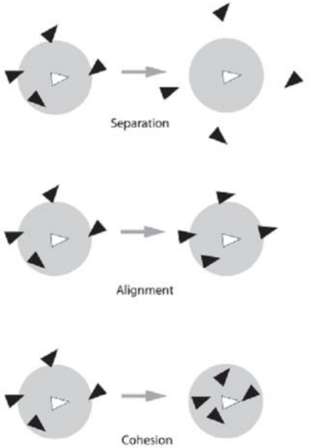
1. When might you precompute paths?
2. This is a single-source, multi-target shortest path algorithm for arbitrary directed graphs with non-negative weights. Question?
3. This is a all-pairs shortest path algorithm.
4. How can a designer allow static paths in a dynamic environment?
5. When will we typically use heuristic search?
6. What is an admissible heuristic?
7. When/Why might we use hierarchical pathing?
8. Does path smoothing work with hierarchical?
9. How might we combat fog-of-war?

Class N-3

1. Steering vs flocking?
2. Steering Family Tree
3. How might we combine behaviors?
4. What three steering mechanisms enable flocking?



Millington Fig 3.29



Buckland Fig 3.16

Class N-2

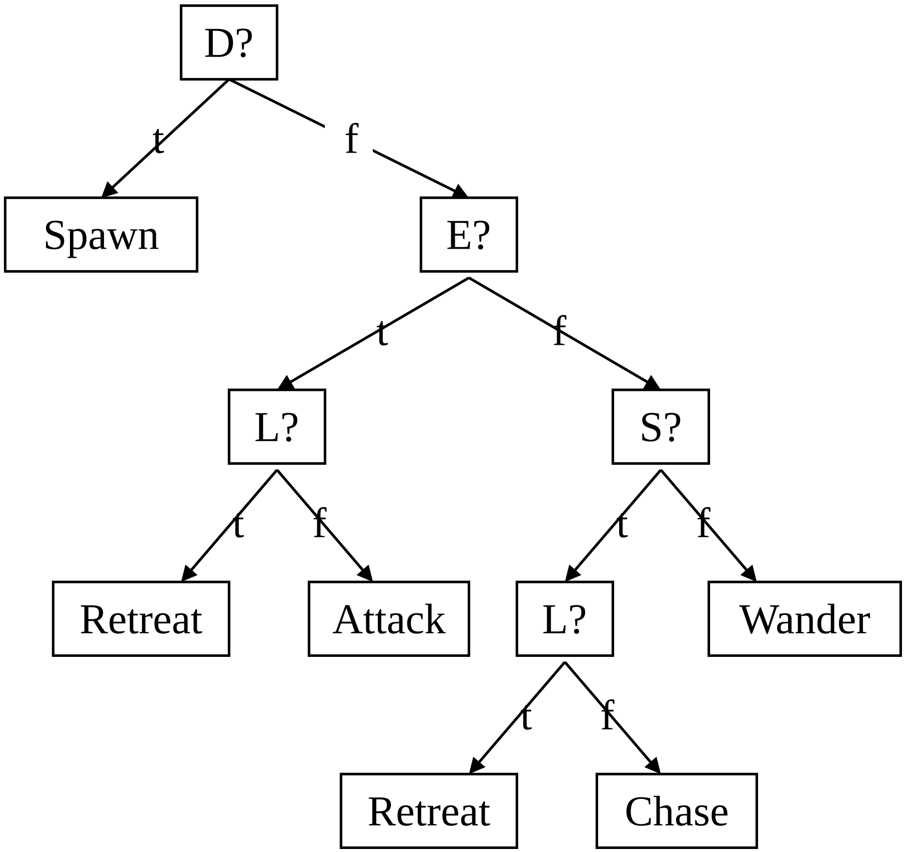
1. How can we describe decision making?
2. What makes FSMs so attractive?
3. What might make us not choose an FSM?
4. Two drawbacks of FSMs, and how to fix?
5. What are the performance dimensions we tend to assess?
6. What are two methods we discussed to learn about changes in the world state?

Class N-1

1. How many outcomes does a d-tree produce?
2. What are advantages of D-Trees?
3. Discuss the effects of tree balance.
4. Must d-trees be a tree?
5. Can d-trees translate into rules? If so how?
6. How can we use d-trees for prediction?
7. What is the notion of overfitting?

Quake D-Tree

- Attributes: E=<t,f> L=<t,f> S=<t,f> D=<t,f>
- Actions: Attack, Retreat, Chase, Spawn, Wander
- Could add additional trees:
  - If I'm attacking, which weapon should I use?
  - If I'm wandering, which way should I go?
  - Much like hierarchical FSMs

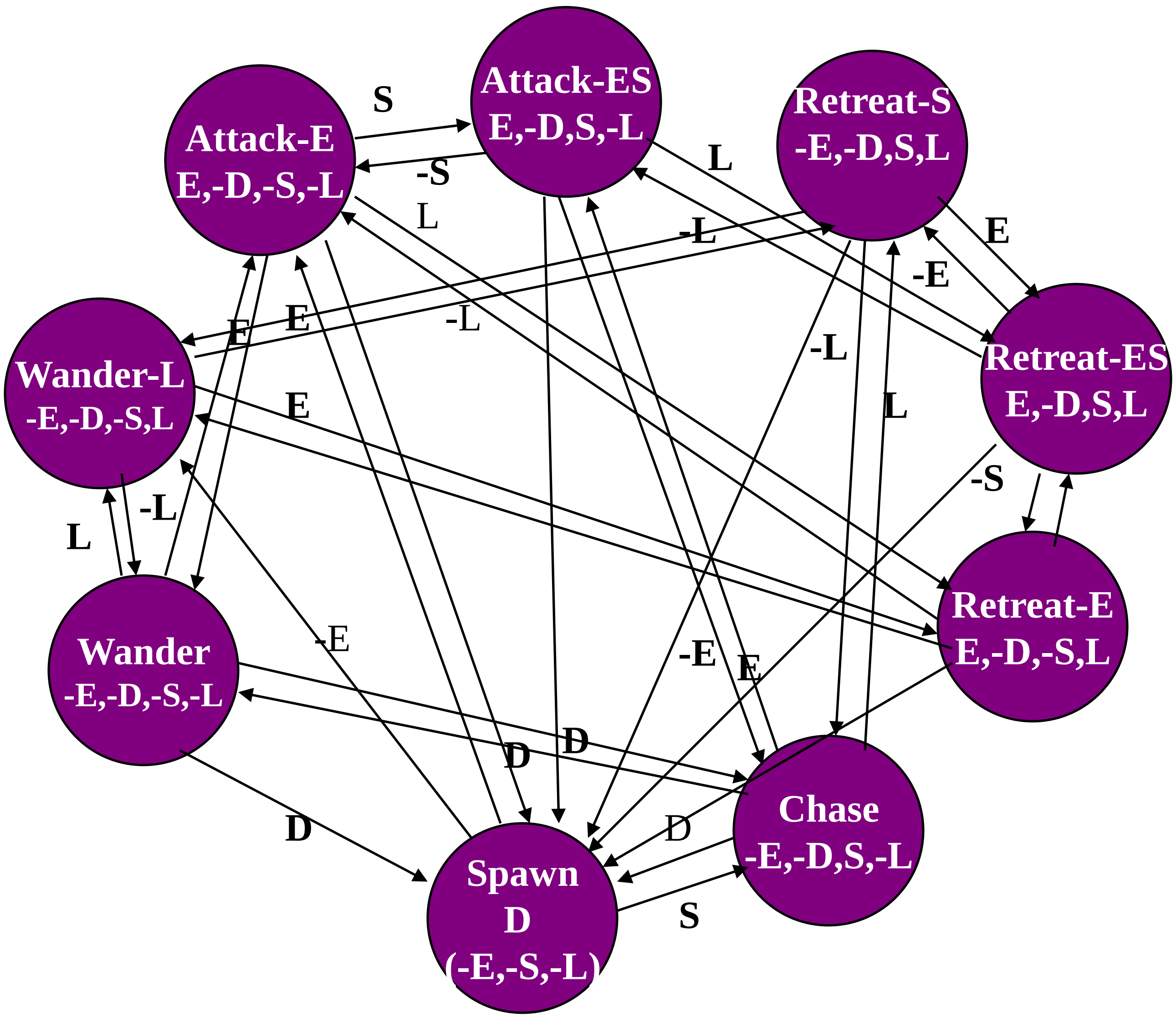


E: Enemy in sight; S: hear a sound;
D: dead; L: Low health

http://research.cs.wisc.edu/graphics/Courses/638-f2001/lectures/cs638-17.ppt.

http://web.eecs.umich.edu/~sugih/courses/eecs494/fall06/lectures/lecture13-gameai.pdf

Quake FSM



OOB

- Decision Making: f(knowledge) → action
  - N+2: Planning
  - N+1: Rule-based Agents, Fuzzy, Markov
  - N: Decision & Behavior Trees (M Ch5.2, 5.4)
  - N-1: Decision & Behavior Trees (M Ch5.2, 5.4)
  - N-2: FSMs
  - N-3: Steering
  - N-4: Graphs, Search, and Movement

Decision Making: Trees

2016-06-09

BEHAVIOR TREES (M CH. 5.4)

Behavior Trees

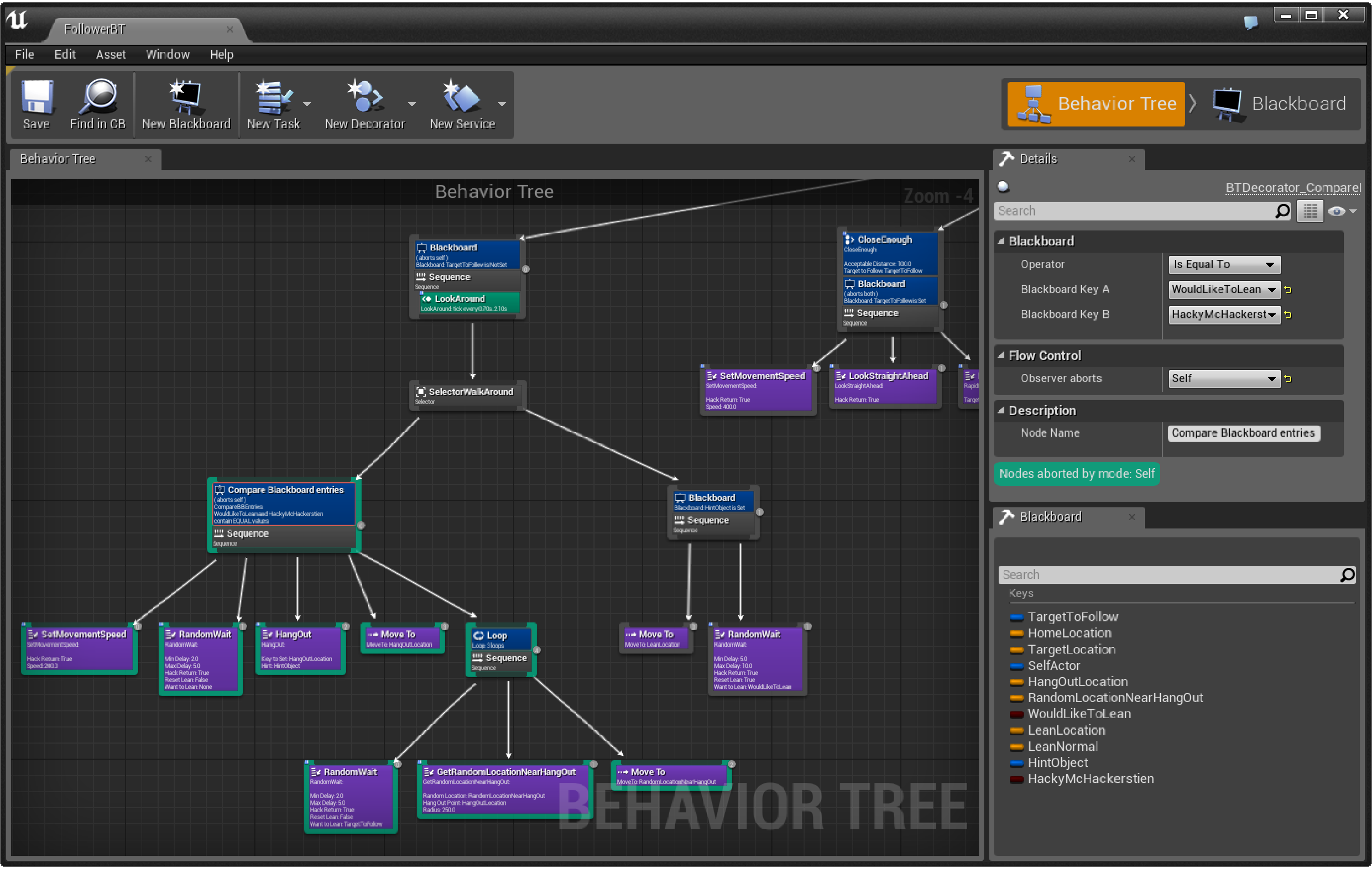
- Very popular/ubiquitous (Bungie's Halo 2 – 2004)
- Synthesis of: HFSM, Scheduling, Planning
- Easy to understand
- Easy for non-programmers to create
- Aren't good in all instances… (stay tuned)
- Composable, self contained
- Instead of *state*, employ *tasks*

BTree Tasks

- Range from looking up variable value to playing animation
- Composed into sub-trees yielding higher-level behaviors
- All task share common interface
- Tasks tend to be self-contained
- Given CPU time to execute, return Success/Failure (error status, need more time)
- Keep each task as small as possible/useful

Behavior Trees

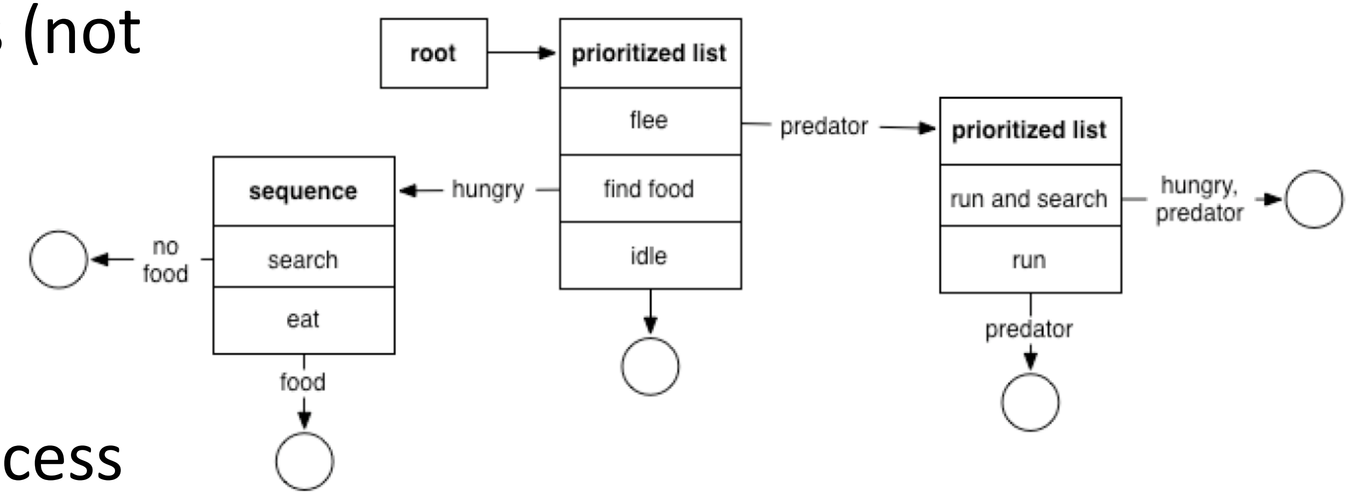
- Simple reactive planning
- Tree of behaviors specify what an agent should do under all circumstances (manually provided)
- Leafs
  - Actions: do something in the world
  - Conditions: test property in the world
- Composite nodes (non-leafs): make a choice/decision
  - (?) Selectors: Prioritized list, (~?) ND
  - (→) Sequence: List, Sequential-looping, (~→) ND



https://docs.unrealengine.com/latest/INT/Engine/AI/BehaviorTrees/index.html

Behavior Tree Structure

- Behavior tree made of connected tasks (not states!)
  - Conditions
  - Actions
  - Composites
- Tasks return success or failure
- Decomposition allows flexibility & easy GUI integration



```
Class Task
{
	children = []

	boolean run ()
	{
		if (execution conditions not met) do {
			return False
		}
		// Do whatever you need to do
		return True or False
	}
}
```

```
Class **Selector** extends Task
{
        boolean run ()
        {
                if (execution conditions not met) do {
                        return False
                }
                for child in children do {
                        if child.run() == True do {
                                return True
                        }
                }
                return False
        }
}
```

```
Class Sequence extends Task
{
        boolean run ()
        {
                if (execution conditions not met) do {
                        return False
                }
                for child in children do {
                        if child.run() == False do {
                                return False
                        }
                }
                return True
        }
}
```

Node Types

- Conditions
- Actions
- Composites

Node Types

- Conditions
  - Test for some game property (e.g. proximity of player to NPC)
  - Each implemented as a task
- Actions
- Composites

Node Types

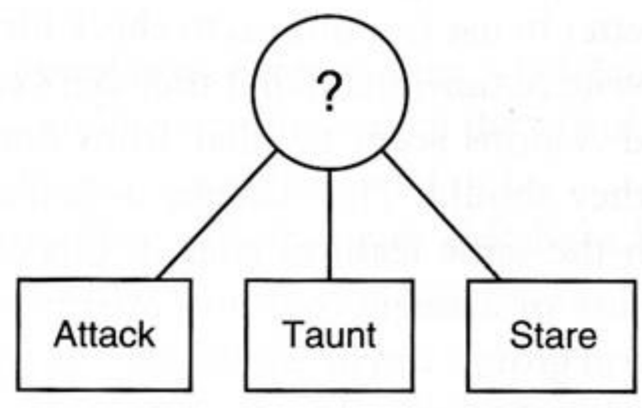
- Conditions
- Actions
  - Alter game state
    - (e.g. play animation, change character internal state, run AI code, play audio sample, etc.)
  - Each is a task
- Composites

Node Types

- Conditions
- Actions
- Composites
  - Differentiates BTs from decision trees
  - Allows for the combination of tasks without concern for what else is in the tree
  - Each is a Task (?)

Composite Nodes: Selector

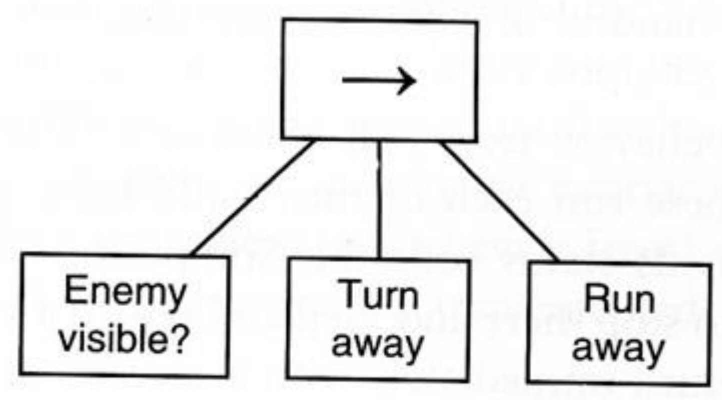
- Selector
  - Run child tasks until one of them succeeds
  - Return failure if all tasks fails



Millington Fig 5.22

Composite Nodes: Sequence

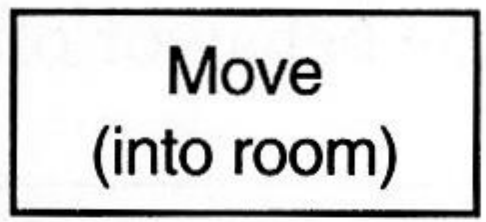
- Selector
- Sequence
  - Series of tasks that all must succeed



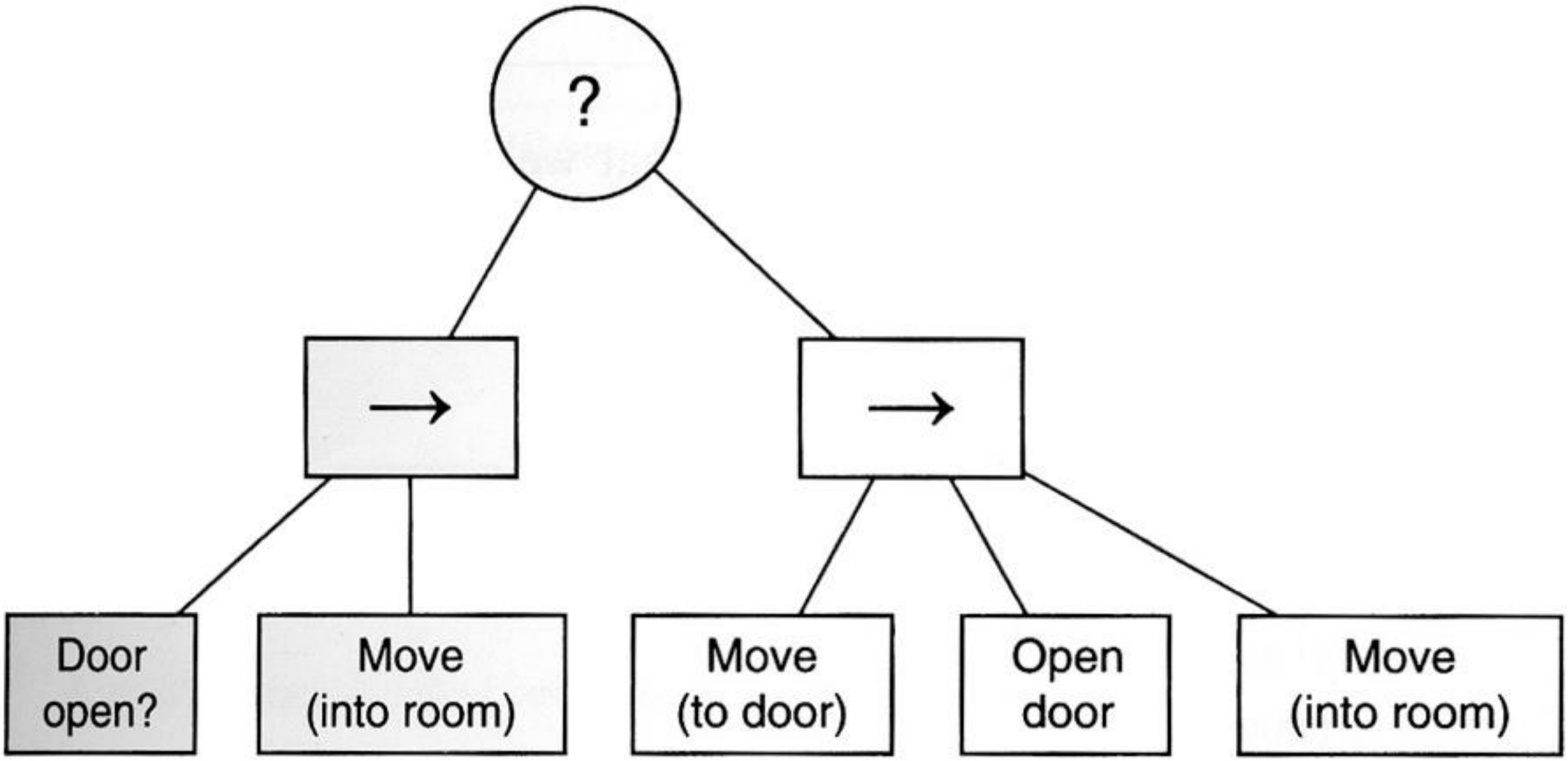
Millington Fig 5.23

Example

Enter room where player is standing.
Player may close the door.



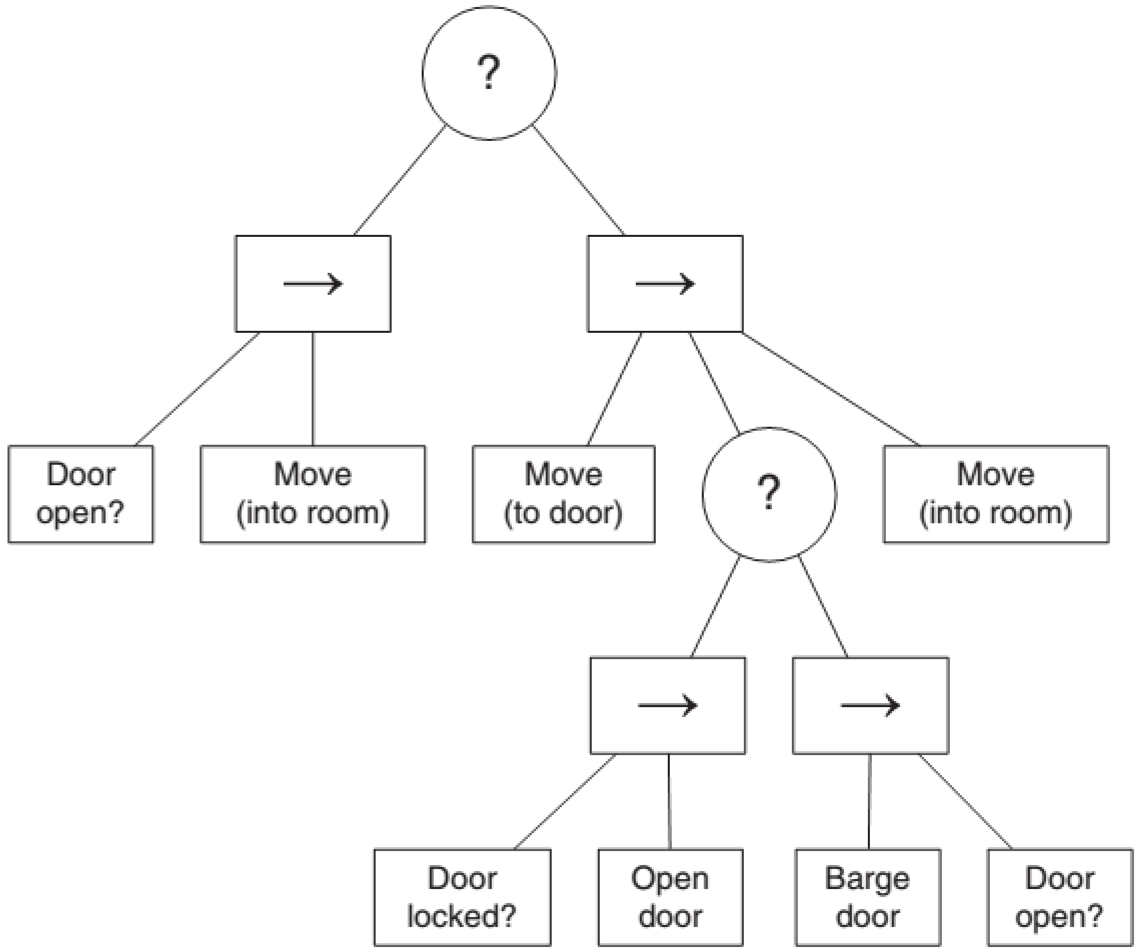
Example



What if the door is locked?

Millington Fig 5.25

Example



Millington Fig 5.27

Non-deterministic Composites

- Strict order == predictable
- We saw partial-orders help this
- Fake partial-order with random shuffle
- 2 new (sub)types of composites
  - ND Selector
  - ND Sequence
  - The original selector/sequence are deterministic (that is, totally ordered)

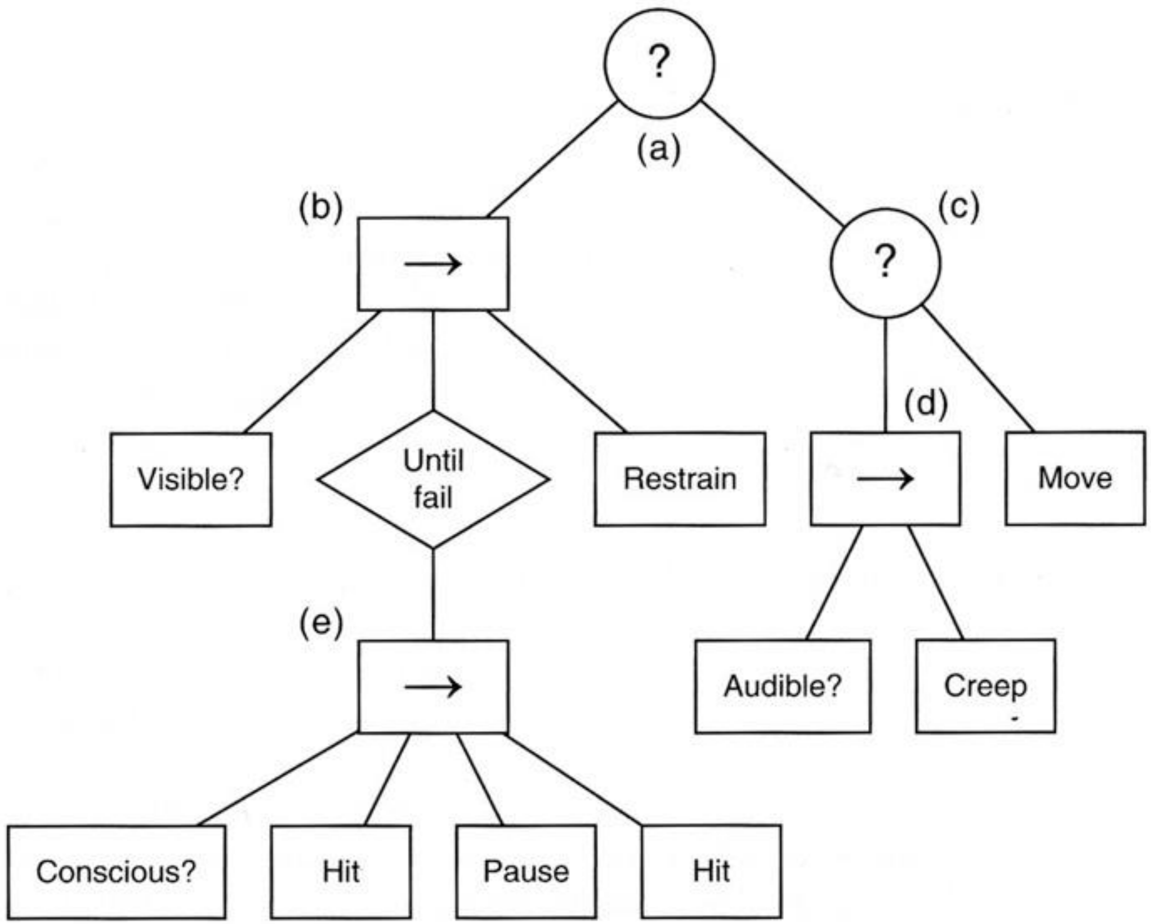
Node summary (so far)

- Conditions
- Action: leaf, alter state of game, move, play animation, etc.
- Composites:
  - Prioritized list: choose subtask, with priority given to certain "questions"
  - Sequence: do all subtasks in order
  - Sequential-looping: sequence, start over when done
  - Probabilistic: randomly choose a subtask
  - One-off: pick one subtask (prioritized or random), but never repeat the choice

4th node type: **Decorators**

- See M CH 5.4.3
- Wraps other composites
- Has a single child task and modifies it in some way
  - Filters (allows child to run (or not))
  - Run Until Fail
  - Inverter
  - Guard Resource (semaphores)

Example

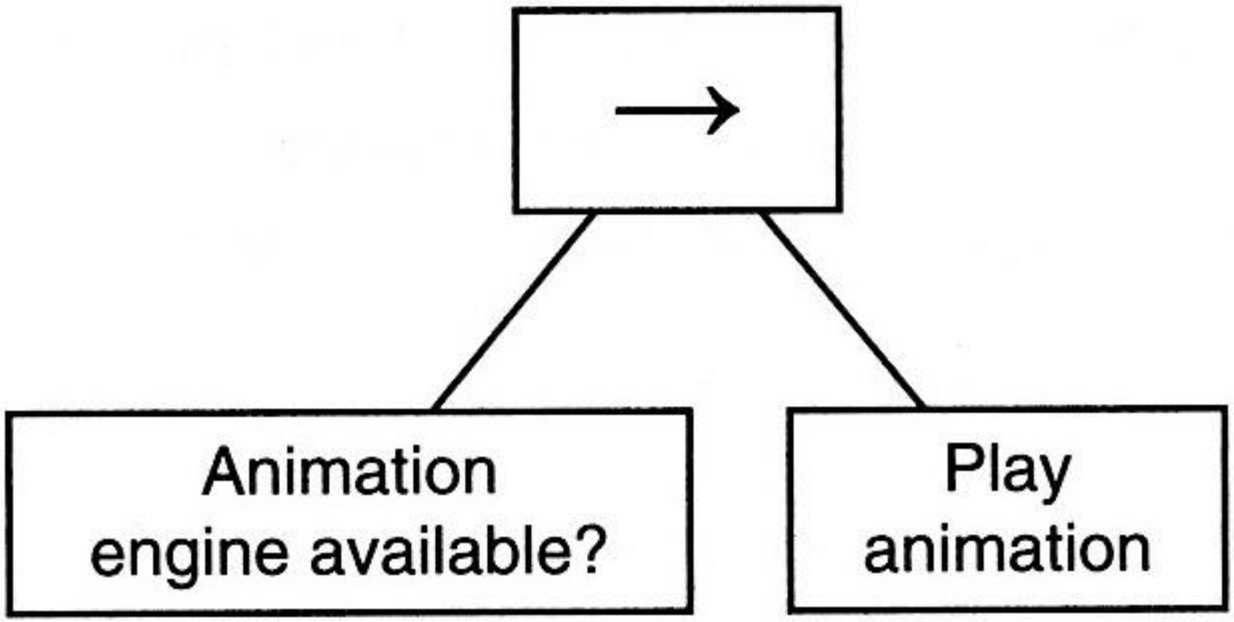


Millington Fig 5.29

Semaphores

- Check for restricted resources
  - Keeps a tally of available resources and number of users
  - e.g. animation engine, pathfinding pool, etc.
- Typically provided in a language library

Guarding Resources

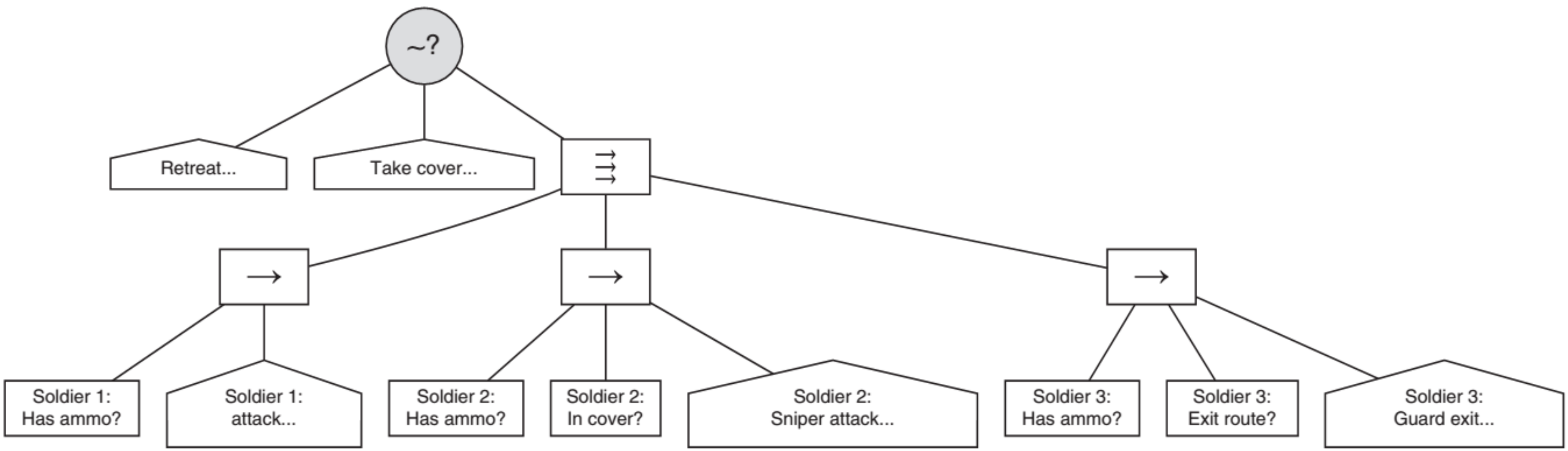


Millington Fig 5.30

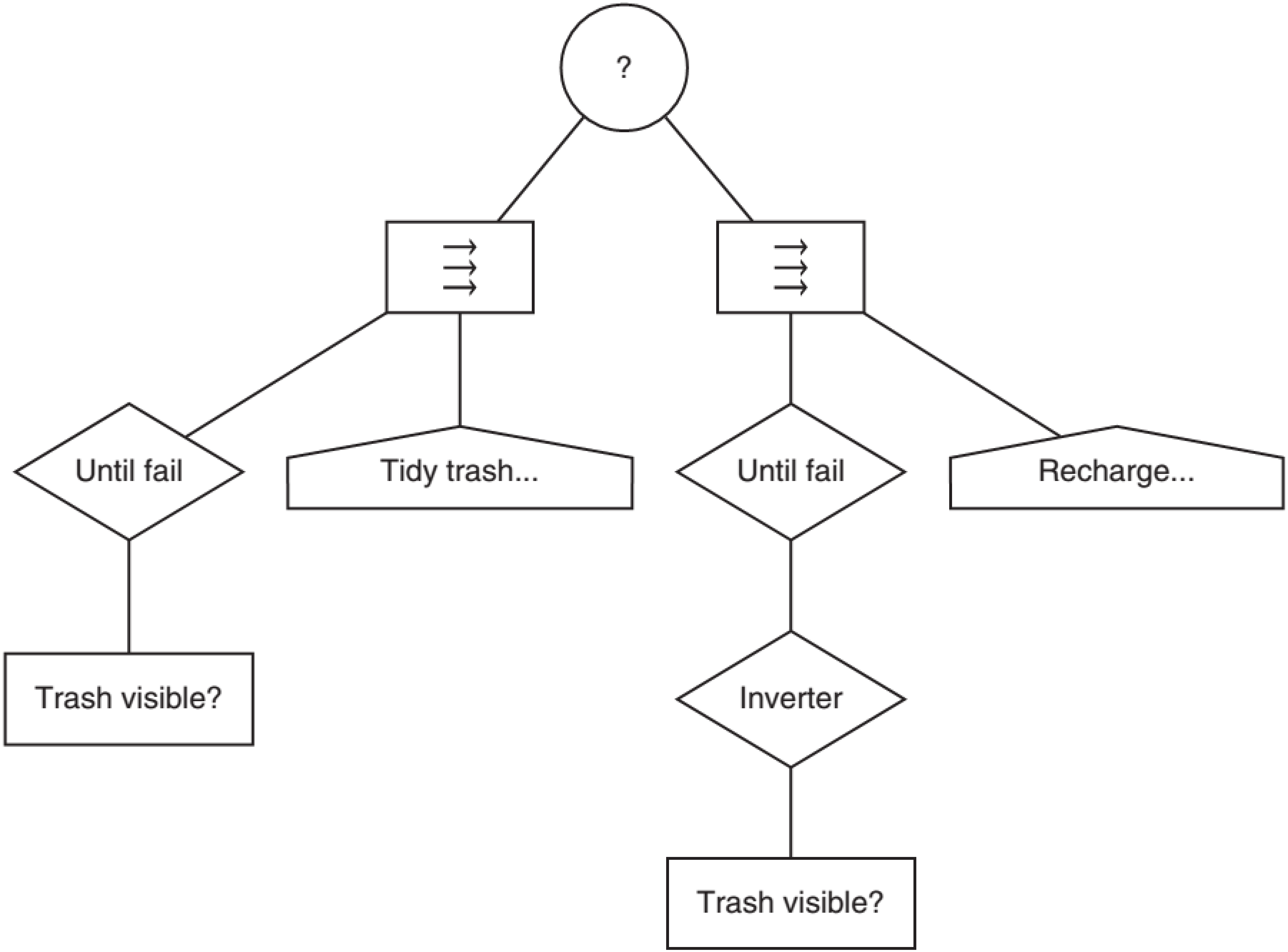
Concurrency & Timing

- Concurrency (tasks run on threads or via multitasking & scheduling algorithms)
  - Essential to make BTs useful
  - Most common practical implementation
  - Millington codebase has example w/ cooperative multitasking
- Blackboard communication for sharing data
- New composite task: Parallel

Why Parallel?

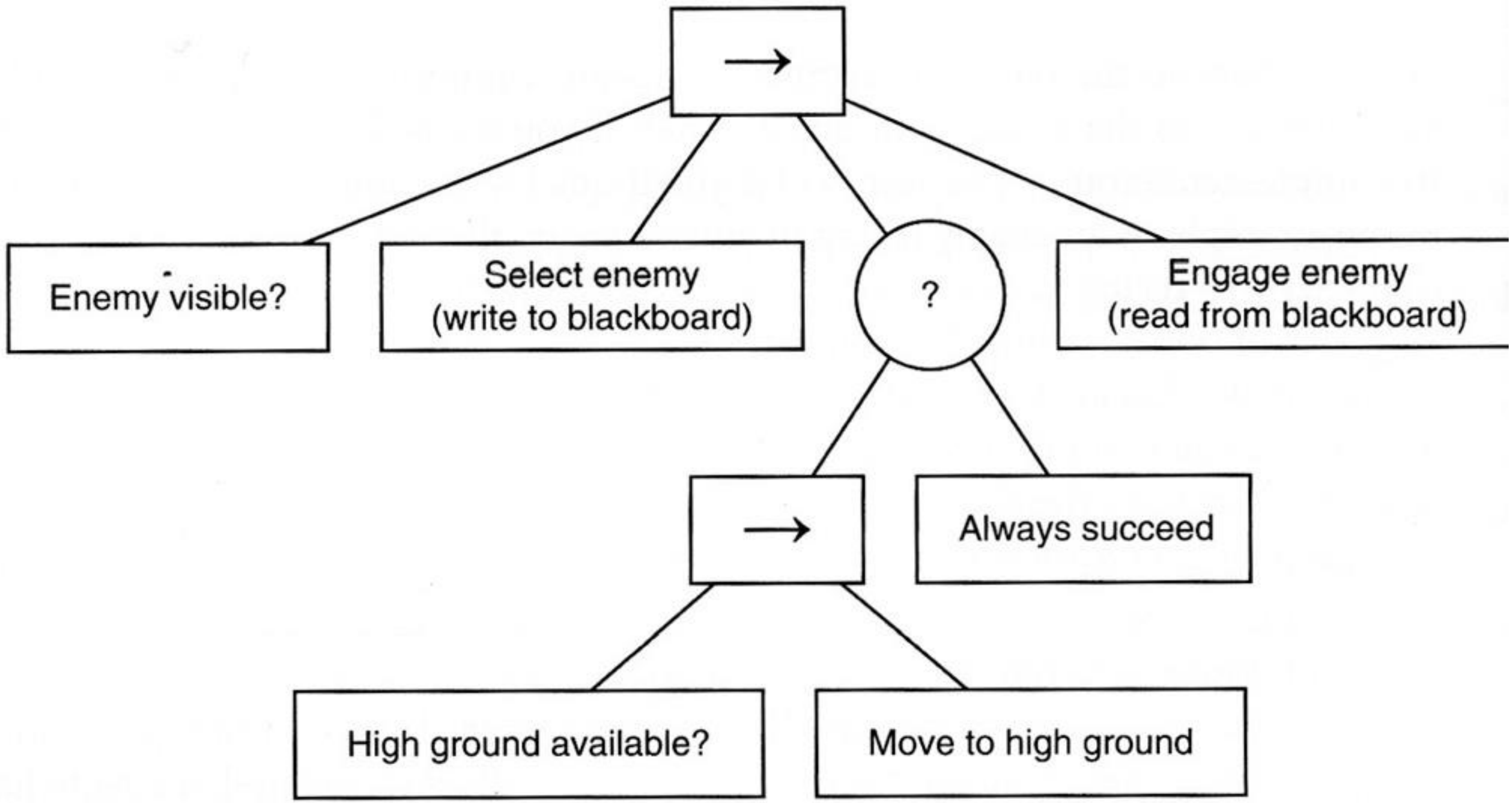


Millington Fig 5.31



Millington Fig 5.34

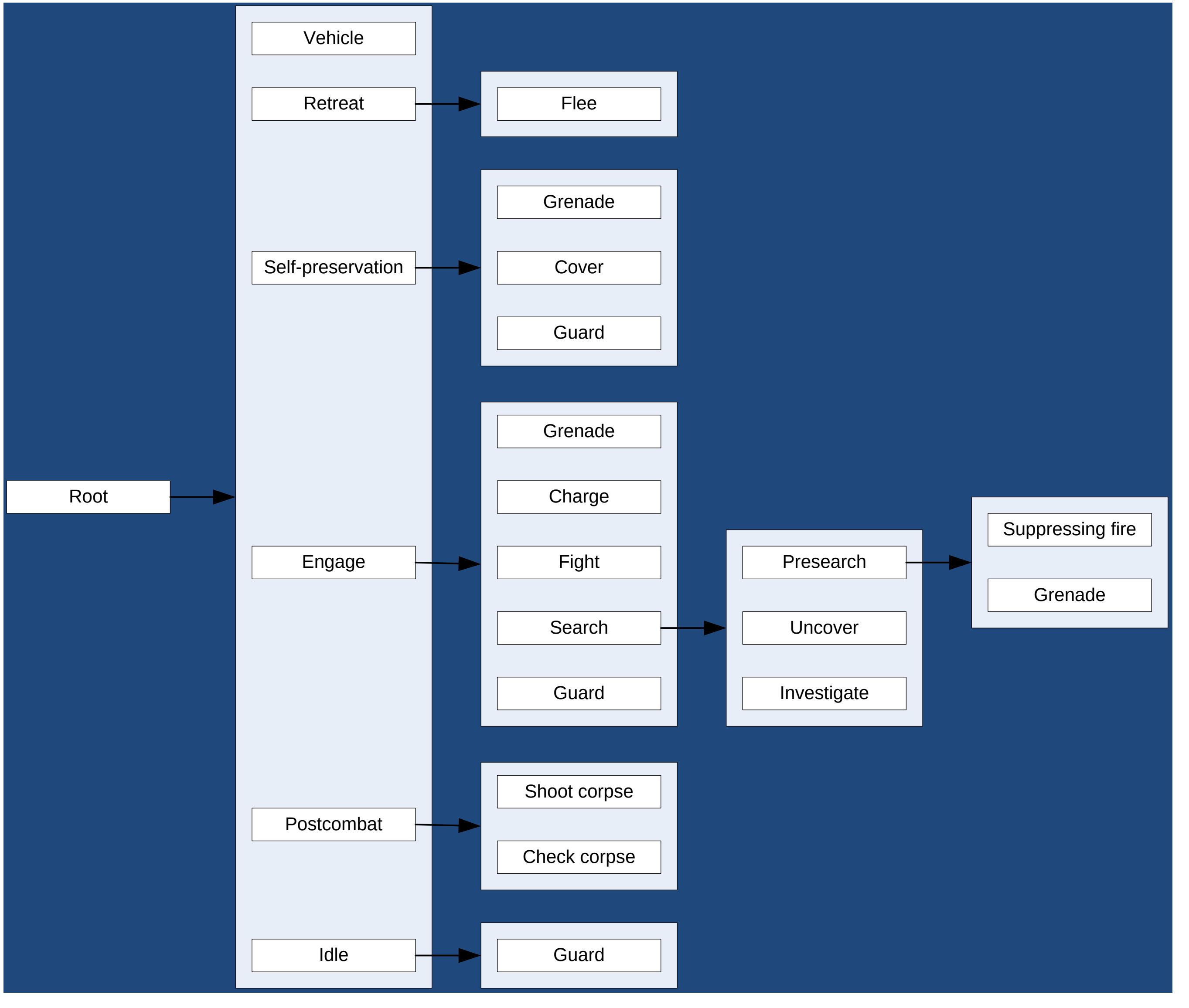
Blackboard Agents



```
Class SelectTarget extends Task
{
        Blackboard bb

        boolean run ()
        {
                character = bb.get('me')
                candidates = enemy_vis_to( character )
                if ( candidates.length > 0)  {
                        targ = biggest_threat( candidates, character )
                        bb.set( 'target', targ )
                        return True
                }
                return False
        }
}
```

BTs in *Halo 2*


Root
Vehicle
Retreat
Self-preservation
Engage
Postcombat
Idle
Flee
Grenade
Cover
Guard
Grenade
Charge
Fight
Search
Guard
Shoot corpse
Check corpse
Guard
Presearch
Uncover
Investigate
Suppressing fire
Grenade

BTs in *Halo 2*

- Determining which behaviors are relevant can be costly (in terms of time)
  - Why? We’re constantly checking relevancy of behaviors that are not actually running
- How can we overcome that?
  - Behavior tagging – Move commonly used checks to decision-time

BT Pros and Cons

- Cons
  - Clunky for state-based behavior
    - That is, changing behavior based on external changes
  - Isn’t really thinking ahead about unique situations
  - Only as good as the designer makes it (just follows the recipes)
- Pros
  - Better when pass/fail of tasks is central
    - Sound familiar? (harder to think about state…)
  - Appearance of goal-driven behavior
  - Multi-step behavior
  - Fast
  - Recover from errors
- Hybrid system may be answer
  - Adds authorial + toolchain burden

Reactive Planning

- Behavior trees implement a simple form of reactive planning
  - Real-time decision making by performing one action every instant

Reactive Planning

- Where a state-action table gives us:

  s_1 --> a_1

  s_2 --> a_2

  …

  we get this from reactive plans:

  s_1 --> a_{11} a_{12} a_{13}…

  s_2 --> a_{21} a_{22}…

  …

Reactive Planning

- Advantages
  - Try things, fail, and fall back
  - Appearance of goal-driven behavior without a formal definition of goals
  - Fast

Reactive Planning

- Advantages
  - Try things, fail, and fall back
  - Appearance of goal-driven behavior without a formal definition of goals
  - Fast
- Disadvantages
  - Can’t really think ahead
  - Only as forward-thinking as the designer makes it

See Also

- Links on previous slides
- AIGPW 4
- http://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php
- http://www.gamasutra.com/blogs/BenWeber/20120308/165151/ABL_versus_Behavior_Trees.php
- Unity or UE4